PROGRAMMER INTERFACE SYSTEM FOR OBJECT DIRECTIONAL COMPUTING SYSTEM, AND STORAGE MEDIUM HAVING OBJECT CODE IN THE SYSTEM

Publication number: JP10124330

Publication date: 1998-05-15

Inventor: DORN KARLHEINZ DIPL-INF; BECKER DETLEF DIPL-ING; QUEHL DIETRICH DIPL-ING

Applicant: SIEMENS AG

Classification:

- international: ***G06F9/46; G06F9/48; G06F9/52;*** **G06F9/46;** (IPC1-7): G06F9/46

- European: G06F9/48C4; G06F9/46M

Application number: JP19970178313 19970703

Priority number(s): US19960675616 19960703

Also published as:

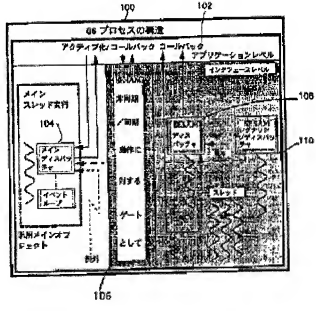
EP0817018 (A2)

US6012081 (A1)

EP0817018 (A3)

Abstract of **JP10124330**

PROBLEM TO BE SOLVED: To obtain a high level frame work independent of an operating system by preparing a programmer interface for concurrency, dispatching and synchronization. SOLUTION: An application programmer acting at an application level 102 in an operating process 100 registers service in a main dispatcher 104 based on the start-up of the process 100. When asynchronization is required in the registered service, asynchronous call back scheduling service is executed in the service by a SESAM dispatcher 108. In this case, both the dispatchers 104, 108 execute a user call back code capable of blocking the dispatcher 104, the executed code is registered in a SESAM and both the dispatchers are informed of the execution of scheduling.



Family list
4 family members for: **JP10124330**
Derived from 3 applications

Back to JP1012

1 **Service and event synchronous/asynchronous manager**
Inventor: DORN KARLHEINZ DIPL-INF (DE); BECKER DETLEF DIPL-ING (DE); (+1)
Applicant: SIEMENS AG (DE)
EC: G06F9/48C4; G06F9/46M
IPC: ***G06F9/46; G06F9/48; G06F9/52*** (+2)
Publication info: **EP0817018 A2** - 1998-01-07
EP0817018 A3 - 2004-05-26

2 **PROGRAMMER INTERFACE SYSTEM FOR OBJECT DIRECTIONAL COMPUTING SYSTEM, AND STORAGE MEDIUM HAVING OBJECT CODE IN THE SYSTEM**
Inventor: DORN KARLHEINZ DIPL-INF; BECKER DETLEF DIPL-ING; (+1)
Applicant: SIEMENS AG
EC: G06F9/48C4; G06F9/46M
IPC: ***G06F9/46; G06F9/48; G06F9/52*** (+2)
Publication info: **JP10124330 A** - 1998-05-15

3 **Service and event synchronous/asynchronous manager**
Inventor: DORN KARLHEINZ (DE); BECKER DETLEF (DE); (+1)
Applicant: SIEMENS AG (DE)
EC: G06F9/48C4; G06F9/46M
IPC: ***G06F9/46; G06F9/48; G06F9/52*** (+2)
Publication info: **US6012081 A** - 2000-01-04

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平10－124330

(43)公開日　平成10年(1998)５月15日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　9/46	３４０	Ｇ０６Ｆ　9/46	３４０Ａ

審査請求　未請求　請求項の数12　ＯＬ　（全 23 頁）

(21)出願番号　特願平9－178313

(22)出願日　平成９年(1997)７月３日

(31)優先権主張番号　０８／６７５６１６

(32)優先日　1996年７月３日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390039413
シーメンス　アクチエンゲゼルシヤフト
ＳＩＥＭＥＮＳ　ＡＫＴＩＥＮＧＥＳＥＬＬＳＣＨＡＦＴ
ドイツ連邦共和国　ベルリン　及び　ミユンヘン　（番地なし）

(72)発明者　カールハインツ　ドルン
ドイツ連邦共和国　カルヒロイト　エアレンシュトラーセ　29

(72)発明者　デトレフ　ベッカー
ドイツ連邦共和国　メーレンドルフ　ヴァッサーヴェルクシュトラーセ　10

(74)代理人　弁理士　矢野　敏雄　（外１名）

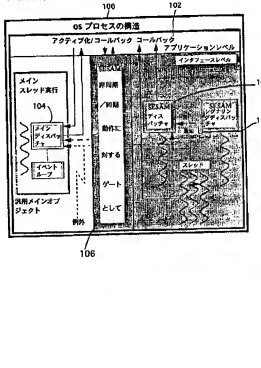
最終頁に続く

(54)【発明の名称】　オブジェクト指向コンピューティングシステムのプログラマインタフェースシステムおよび該システムをもつオブジェクトコードを有する記憶媒体

(57)【要約】

【課題】　オブジェクト指向コンピューティングシステムにおいて、コンカレンシ、ディスパッチングならびに同期化のためのプログラマインタフェースを用意するサービスおよびイベント同期／非同期マネージャを提供する。

【解決手段】　ダイナミックスロットと、同期タイマスロットと、非同期タイマスロットと、例外スロットと、プログラマのイベントに対し外部イベントの通知を行うスロットと、スレッドディスパッチャと、メインディスパッチャと、シグナリングディスパッチャと、スレッドマネージャと、メッセージブロックメモリと、ユーザイベントに対しエラー外部イベント通知を戻すアクティビティ識別子のリストと、アクティビティ識別子をスロット識別子にマッピングするためのアドミニストレータと、ブロックされたスレッドのためのキューとが設けられている。



【特許請求の範囲】
【請求項１】　コンピュータプラットフォームにおけるオブジェクト指向コンピューティングシステムのプログラマインタフェースシステムにおいて、
ａ）プログラマの依頼する関数を非同期に実行させる少なくとも１つのダイナミックスロットと、
ｂ）少なくとも１つの同期タイマスロットと、
ｃ）少なくとも１つの非同期タイマスロットと、
ｄ）プログラマの定義したシステム例外コールバックを処理するための少なくとも１つの例外スロットと、
ｅ）プログラマのイベントに対し外部イベントの通知を行う少なくとも１つのスロットと、
ｆ）スレッドディスパッチャと、
ｇ）メインディスパッチャと、
ｈ）シグナリングディスパッチャと、
ｉ）スロットにより要求されるスレッドの実行を管理するスレッドマネージャと、
ｊ）メッセージブロックを格納するためのメッセージブロックメモリと、
ｋ）ユーザイベントに対しエラー外部イベント通知を戻すアクティビティ識別子のリストと、
ｌ）アクティビティ識別子をスロット識別子にマッピングするためのアドミニストレータと、
ｍ）ブロックされたスレッドのためのキュー、とが設けられていることを特徴とする、
コンピュータプラットフォームにおけるオブジェクト指向コンピューティングシステムのプログラマインタフェースシステム。
【請求項２】　複数のダイナミックスロットが設けられており、該ダイナミックスロットは、
シグナリングがなくまだ使用されていないときには初期状態をとり、
シグナリングがなく使用中のときにはキューイング状態をとり、
シグナリングがなく使用中であって依頼された関数が実行されているときにスタート状態をとり、
シグナリングがあり使用中であって依頼された関数の実行が完了しているときには終了状態をとり、
シグナリングがあり使用中であって依頼された関数が実行されているときにはコールバックアクティブ状態をとり、
シグナリングがありもはや使用されていなければアイドル状態をとり、
ここでシグナリングあり状態は、プログラマの依頼した関数が実行されたときにとられるものである、ように構成されている、
請求項１記載のオブジェクト指向コンピューティングシステムのプログラマインタフェースシステム。
【請求項３】　オペレーティングシステムのディスパッチャへのポインタのかたちでメインディスパッチャが設けられている、請求項１記載のオブジェクト指向コンピューティングシステムのプログラマインタフェースシステム。
【請求項４】　オブジェクト指向コンピューティングシステムのプログラマインタフェースシステムにおいて、集中的にスレッドをディスパッチングし、ランタイムにコンフィグレーション可能なイベントハンドラの引き渡しを行う手段と、
イベントハンドラの実行をインタラプトする手段と、
イベントハンドラにおいてイベント待機をブロックする手段と、
インタフェースシステムの外部で発生するイベントに関して非同期で関数を実行させる手段と、
実行中の他のイベントハンドラに対しパラレルに非同期タイマハンドラを実行させる手段とが設けられていることを特徴とする、
オブジェクト指向コンピュータシステムのプログラマインタフェースシステム。
【請求項５】　オブジェクト指向コンピューティングシステムのプログラマインタフェースシステムにおいて、コールバックにより駆動されるシステムをインタラプトする手段が設けられており、該手段はコールバック自体をインタラプトし、
非同期ディスパッチャと、
同期ディスパッチャと、
非同期のシグナリングディスパッチャと、
スレッドマネージャと、
種々の型の複数のスロットが設けられており、該スロットはそれらに割り当てられたアクティビティを管理するために用いられることを特徴とする、
オブジェクト指向コンピューティングシステムのプログラマインタフェースシステム。
【請求項６】　前記の種々の型の複数のスロットは、
アプリケーションプログラマのために非同期を管理する少なくとも１つのダイナミックスロットと、
同期タイマを管理する少なくとも１つの同期タイマスロットと、
非同期タイマを管理する少なくとも１つの非同期タイマスロットと、
ユーザにより定義されたシステム例外コールバックを管理する少なくとも１つの例外スロットと、
ユーザイベントのための waitFor()...() 機能を管理する少なくとも１つの汎用スロット、とにより構成されている、請求項５記載のオブジェクト指向コンピューティングシステムのプログラマインタフェースシステム。
【請求項７】　コンピュータプラットフォームにおけるオブジェクト指向コンピューティングシステムのプログラマインタフェースシステムをもつオブジェクト指向コードを有する記憶媒体において、
ａ）プログラマの依頼する関数を非同期に実行させる少

なくとも１つのダイナミックスロットと、
ｂ）少なくとも１つの同期タイマスロットと、
ｃ）少なくとも１つの非同期タイマスロットと、
ｄ）プログラマの定義したシステム例外コールバックを処理するための少なくとも１つの例外スロットと、
ｅ）プログラマのイベントに対し外部イベントの通知を行う少なくとも１つのスロットと、
ｆ）スレッドディスパッチャと、
ｇ）メインディスパッチャと、
ｈ）シグナリングディスパッチャと、
ｉ）スロットにより要求されるスレッドの実行を管理するスレッドマネージャと、
ｊ）メッセージブロックを格納するためのメッセージブロックメモリと、
ｋ）ユーザイベントに対しエラー外部イベント通知を戻すアクティビティ識別子のリストと、
ｌ）アクティビティ識別子をスロット識別子にマッピングするためのアドミニストレータと、
ｍ）ブロックされたスレッドのためのキュー、とが設けられていることを特徴とする、
コンピュータプラットフォームにおけるオブジェクト指向コンピューティングシステムのプログラマインタフェースシステムをもつオブジェクト指向コードを有する記憶媒体。
【請求項８】　複数のダイナミックスロットが設けられており、該ダイナミックスロットは、
シグナリングがなくまだ使用されていないときには初期状態をとり、
シグナリングがなく使用中のときにはキューイング状態をとり、
シグナリングがなく使用中であって依頼された関数が実行されているときにスタート状態をとり、
シグナリングがあり使用中であって依頼された関数の実行が完了しているときには終了状態をとり、
シグナリングがあり使用中であって依頼された関数が実行されているときにはコールバックアクティブ状態をとり、
シグナリングがありもはや使用されていなければアイドル状態をとり、
ここでシグナリングあり状態は、プログラマの依頼した関数が実行されたときにとられるものである、ように構成されている、
請求項７記載の記憶媒体。
【請求項９】　オペレーティングシステムのディスパッチャへのポインタのかたちでメインディスパッチャが設けられている、請求項７記載の記憶媒体。
【請求項１０】　オブジェクト指向コンピューティングシステムのプログラマインタフェースシステムをもつオブジェクト指向コードを有する記憶媒体において、
集中的にスレッドをディスパッチングし、ランタイムにコンフィグレーション可能なイベントハンドラの引き渡しを行う手段と、
イベントハンドラの実行をインタラプトする手段と、
イベントハンドラにおいてイベント待機をブロックする手段と、
インタフェースシステムの外部で発生するイベントに関して非同期で関数を実行させる手段と、
実行中の他のイベントハンドラに対しパラレルに非同期タイマハンドラを実行させる手段とが設けられていることを特徴とする、
オブジェクト指向コンピューティングシステムのプログラマインタフェースシステムをもつオブジェクト指向コードを有する記憶媒体。
【請求項１１】　オブジェクト指向コンピューティングシステムのプログラマインタフェースシステムをもつオブジェクト指向コードを有する記憶媒体において、
コールバックにより駆動されるシステムをインタラプトする手段が設けられており、該手段はコールバック自体をインタラプトし、
非同期ディスパッチャと、
同期ディスパッチャと、
非同期のシグナリングディスパッチャと、
スレッドマネージャと、
種々の型の複数のスロットが設けられており、該スロットはそれらに割り当てられたアクティビティを管理するために用いられることを特徴とする、
オブジェクト指向コンピューティングシステムのプログラマインタフェースシステムをもつオブジェクト指向コードを有する記憶媒体。
【請求項１２】　前記の種々の型の複数のスロットは、
アプリケーションプログラマのために非同期を管理する少なくとも１つのダイナミックスロットと、
同期タイマを管理する少なくとも１つの同期タイマスロットと、
非同期タイマを管理する少なくとも１つの非同期タイマスロットと、
ユーザにより定義されたシステム例外コールバックを管理する少なくとも１つの例外スロットと、
ユーザイベントのための waitFor()...() 機能を管理する少なくとも１つの汎用スロット、とにより構成されている、
請求項１１記載の記憶媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、コンピュータプラットフォームにおけるオブジェクト指向コンピューティングシステムのプログラマインタフェースシステム、および該システムをもつオブジェクトコードを有する記憶媒体に関する。
【０００２】

【従来の技術】アメリカ合衆国特許第5，499，365号公報に示されているように、オブジェクト指向プログラミングシステムおよびプロセス（"オブジェクト指向コンピューティング環境"とも称する）は多くの研究テーマや関心の対象となってきている。当業者によく知られているように、オブジェクト指向プログラミングシステムは多数の"オブジェクト"により構成されている。1つのオブジェクトは、"フレーム"とも呼ばれるデータ構造と、このデータ構造にアクセスでき"メソッド"とも呼ばれるオペレーションないしファンクション（関数）のセットである。フレームは複数の"スロット"を有することができ、それらのスロットの各々にはスロット内のデータの"属性"が含まれている。この属性は（整数や文字列のような）プリミティブとすることができるし、あるいは他のオブジェクトへのポインタとなるオブジェクトリファレンスとすることができる。同一のデータ構造ならびに共通の特性をもつオブジェクトをグループにまとめることができ、それをひとまとめにして"クラス"として定義できる。

【０００３】オブジェクトにおける定義された各クラスは通常、複数の"インスタンス"において明示されることになる。各インスタンスには、オブジェクトの特定の実例に対する固有のデータ構造が含まれている。オブジェクト指向コンピューティング環境の場合、オブジェクトをリクエストすることによりデータが処理され、オブジェクトに"メッセージ"を送ってそのメソッドのうちの1つを実行させる。それを受け取ったオブジェクトはメッセージに対して応答し、メッセージネームを実行するメソッドを選択し、そのメソッドを指定されたインスタンスにおいて実行し、メソッドの結果といっしょに呼び出し高位レベルルーチンへ制御を戻す。クラス、オブジェクトおよびインスタンス間の関係は因襲的に、"ビルトタイム"中またはオブジェクト指向コンピューティング環境の生成中に確立されたものであり、つまりオブジェクト指向コンピューティング環境の"ランタイム"すなわち実行よりも前に確立されている。

【０００４】既述のクラス、オブジェクトおよびインスタンスの関係に加えて、2つまたはそれ以上のクラス間に継承関係も存在しており、これは第1のクラスが第2のクラスの"親"とみなすことができ、第2のクラスを第1のクラスの"子"とみなすことができるようなものである。換言すれば、第1のクラスは第2のクラスの先祖であり、第2のクラスは第1のクラスの子孫であり、第2のクラス（つまり子孫）は第1のクラス（つまり先祖）を継承するものであるといわれるようなことである。子クラスのデータ構造は親クラスの属性すべてを有している。

【０００５】したがってオブジェクト指向システムはこれまで、オブジェクトの"バージョン"を認識してきた。あるオブジェクトのバージョンは、異なる時点でのこのオブジェクトと同じデータである。たとえば"進行中の作業"に係わるオブジェクトは、完了し承認済みの作業に係わる同じオブジェクトデータの異なるバージョンである。したがって多くの適用事例において、様々な時点で存在しているようなデータの履歴レコードが必要とされる。このため、種々のバージョンのオブジェクトが必要とされる。

【０００６】E.W.Dijkstra, The STructure of "THE" Multi programming System, Communications of the ACM, Vol.11, No.5, May 1968, pp. 341-346, および C.A.R.Hoare, Monitors: Operating Systems Structuring Concepts, Communications ofthe ACM, Vol. 17, No. 10, October, 1974, pp. 549-557 には、全般的な背景を示す2つの論文が示されている。最初の論文にはプリミティブを用いた同期化方法について述べられていて、セマフォの使用が説明されている一方、後者の論文にはオペレーティングシステムの構築手法として Brinch-Hansen のモニタのコンセプトが明らかにされている。特に Hoare の論文にはプロセスのための同期化のフォームが紹介されていて、セマフォに基づく可能なインプリメント手法が述べられており、証明規則ならびに実例が示されている。

【０００７】・スレッド（Thread）
1つのプロセス内の並列実行ユニット。モニタは強制的な逐次化により、同時に実行している複数のスレッドの並行なアクセスを同期させ、それらのスレッドはすべてモニタを通して保護されている1つのオブジェクトの関数をコールする。

【０００８】・同期プリミティブ（Synchronizations-Primitive）
並列アクティビティのレシプロカルジャスティフィケーションのためのオペレーティングシステムの手段。

【０００９】・セマフォ（Senaphore）
並列アクティビティのための同期プリミティブ。

【００１０】・Ｍｕｔｅｘ
並列アクティビティのための特別な同期プリミティブであり、相互排除の目的でクリティカルなコードレンジを有する。

【００１１】・コンディション・キュー（Condition Queue）
ある所定のコンディションを参照する並列アクティビティのためのイベント・キュー

・ゲートロック（Gate Lock）
各エントリ関数のため、オブジェクトの保護のため、1度にただ1つの並列アクティビティに対してしかオブジェクトのエントリルーチンの使用を許可しないためのモニタのｍｕｔｅｘである。

【００１２】・長期間スケジューリング（Long Term Scheduling）
並列アクティビティのためのコンディション・キューま

たはイベント待ちキュー内における１つの並列アクティビティの長期間ディレイである。

【００１３】・ブローカ（Broker）

ディストリビュータ。

【００１４】さらに以下の略語がここで用いられる可能性がある：

AFM

Asynchronous Function Manager（非同期ファンクションマネージャ）

SESAM

Service & Event Synchronous Asynchronous Manager（サービスおよびイベント同期非同期マネージャ）

PAL

Programmabel Area Logic（プログラム可能なエリアロジック）

API

Application Programmers Interface（アプリケーションプログラマーインタフェース）

IDL

Interface Definition Language（インタフェース定義言語）

ATOMIC

Asynchron Transport Optimizing observer-pattern-like system supporingseveral Modes(client/server-push/pull) for an IDL-less Communication subsystem（ＩＤＬを用いない通信サブシステムのための複数のモード（クライアント／サーバープッシュ／プル）をサポートする非同期転送最適化オブザーバーパターンライクシステム）

XDR

External Data Representation（外部データ表現）

I/O

Input/Output（入／出力）

IPC

Inter Process Communication（プロセス間通信）

CSA

Common Software Architecture(a Siemens AG computing system convention、共通ソフトウェアアーキテクチャ）

SW

Software（ソフトウェア）

【００１５】

【発明が解決しようとする課題】本発明の課題は、オブジェクト指向コンピューティングシステムにおいて、コンカレンシ、ディスパッチングならびに関化のためのプログラマインタフェースを用意するサービスおよびイベント同期／非同期マネージャ（ＳＥＳＡＭ）を提供することにある。

【００１６】

【課題を解決するための手段】本発明によればこの課題は、ａ）プログラマの依頼する関数を非同期に実行させる少なくとも１つのダイナミックスロットと、

ｂ）少なくとも１つの同期タイマスロットと、

ｃ）少なくとも１つの非同期タイマスロットと、

ｄ）プログラマの定義したシステム例外コールバックを処理するための少なくとも１つの例外スロットと、

ｅ）プログラマのイベントに対し外部イベントの通知を行う少なくとも１つのスロットと、

ｆ）スレッドディスパッチャと、

ｇ）メインディスパッチャと、

ｈ）シグナリングディスパッチャと、

ｉ）スロットにより要求されるスレッドの実行を管理するスレッドマネージャと、

ｊ）メッセージブロックを格納するためのメッセージブロックメモリと、

ｋ）ユーザイベントに対しエラー外部イベント通知を戻すアクティビティ識別子のリストと、

ｌ）アクティビティ識別子をスロット識別子にマッピングするためのアドミニストレータと、

ｍ）ブロックされたスレッドのためのキュー、

とが設けられていることにより解決される。

【００１７】さらに本発明によれば上記の課題は、請求項４、請求項５、請求項７、請求項１０および請求項１１に記載の構成により解決される。

【００１８】

【発明の実施の形態】したがって、プログラマはローレベルのスレッドの詳細に煩わされることがなくなる。このインタフェースは同期／非同期のファンクションへのゲートとして用いられる。

【００１９】ＳＥＳＡＭの役割は、オペレーティングシステムに依存しないハイレベルのフレームワークを提供することである。

【００２０】また、ＳＥＳＡＭの役割は、非同期のためのスレッドの使用法と、非同期のためのものではないオペレーティングシステムからの独立性を提供することである。

【００２１】さらにＳＥＳＡＭの役割は、きわめてハイレベルの"ｗａｉｔ－ｆｏｒ－ｅｖｅｎｔ"インタフェースをサポートすることである。このインタフェースにより単一のハイレベルの"アプリケーション－メインループ"は、すべてのローレベルのオペレーティングシステムに依存するイベントディスパッチングならびにハイレベルのアプリケーションイベントディスパッチングを獲得できるようになる。

【００２２】ＳＥＳＡＭの役割は、アプリケーションおよびオペレーティングシステムについてのポータビリティである。

【００２３】さらにＳＥＳＡＭの役割は、ただ１つの同種の解決手段においてアクティブなオブジェクトパターンもパッシブなオブジェクトパターンもサポートする目

的で、従来の非同期機能の同期化も行うことである。

【００２４】１つの実施形態によれば、複数のダイナミックスロットが設けられており、該ダイナミックスロットは、シグナリングがなくまだ使用されていないときには初期状態をとり、シグナリングがなく使用中のときにはキューイング状態をとり、シグナリングがなく使用中であって依頼された関数が実行されているときにスタート状態をとり、シグナリングがあり使用中であって依頼された関数の実行が完了しているときには終了状態をとり、シグナリングがあり使用中であって依頼された関数が実行されているときにはコールバックアクティブ（ｂ Active）状態をとり、シグナリングがありもはや使用されていなければアイドル状態をとり、ここでシグナリングあり状態は、プログラマの依頼した関数が実行されたときにとられるものである、ように構成されている。

【００２５】また、１つの実施形態によれば、オペレーティングシステムのディスパッチャへのポインタのかたちでメインディスパッチャが設けられている。

【００２６】さらに１つの実施形態によれば、オブジェクト指向コンピューティングシステムのプログラマインタフェースシステムにおいて、集中的にスレッドをディスパッチングし、ランタイムにコンフィグレーション可能なイベントハンドラの引き渡しを行う手段と、イベントハンドラの実行をインタラプトする手段と、イベントハンドラにおいてイベント待機をブロックする手段と、インタフェースシステムの外部で発生するイベントに関して非同期で関数を実行させる手段と、実行中の他のイベントハンドラに対しパラレルに非同期タイマハンドラを実行させる手段とが設けられている。

【００２７】また、１つの実施形態によれば、オブジェクト指向コンピューティングシステムのプログラマインタフェースシステムにおいて、コールバックにより駆動されるシステムをインタラプトする手段が設けられており、該手段はコールバック自体をインタラプトし、非同期ディスパッチャと、同期ディスパッチャと、非同期のシグナリングディスパッチャと、スレッドマネージャと、種々の型の複数のスロットが設けられており、該スロットはそれらに割り当てられたアクティビティを管理するために用いられる。

【００２８】さらに１つの実施形態によれば、前記の種々の型の複数のスロットは、アプリケーションプログラマのために非同期を管理する少なくとも１つのダイナミックスロットと、同期タイマを管理する少なくとも１つの同期タイマスロットと、非同期タイマを管理する少なくとも１つの非同期タイマスロットと、ユーザにより定義されたシステム例外コールバックを管理する少なくとも１つの例外スロットと、ユーザイベントのための waitFor()...() 機能を管理する少なくとも１つの汎用スロットとにより構成されている。

【００２９】次に、図面を参照しながら本発明について詳細に説明する。

【００３０】

【実施例】アメリカ合衆国特許第５，４９９，３６５号に示されているようにオブジェクト指向コンピューティング環境では、データを含むオブジェクトへアクションリクエストメッセージが送られることによりワークが実行される。オブジェクトはまえもって定められたメソッドに従って、データに対しリクエストされたアクションを実行する。オブジェクトはオブジェクトクラスにまとめることができ、オブジェクトクラスによりデータの型と意味ならびにオブジェクトが受け取るアクションリクエスト（メッセージ）が定義される。データが含まれている個々のオブジェクトはクラスのインスタンスと呼ばれる。

【００３１】オブジェクトクラスは他のクラスのサブクラスとして定義できる。サブクラスは親クラスのデータ特性およびメソッドのすべてを継承する。サブクラスは付加的なデータおよびメソッドの追加を行うことができ、親クラスのいかなるデータエレメントも無効にしたり再定義したりすることができる。

【００３２】図１、図２および図３はアメリカ合衆国特許第５，４９９，３６５号から転載したものである。それらの図に関する以下の説明は上記の特許をもとにしている。

【００３３】まずはじめに図３を参照すると、この図にはハードウェアおよびソフトウェアの環境が示されている。図３には、１つまたは複数のコンピュータプラットフォーム１２上で動作するオブジェクト指向コンピューティング環境１１が示されている。この場合、当業者であれば理解できるようにコンピュータプラットフォーム１２は典型的には、中央処理ユニット（ＣＰＵ）１４、メインメモリ１５および入／出力（Ｉ／Ｏ）インタフェース１６というようなコンピュータハードウェアユニット１３を有しており、さらにディスプレイ端末２１、キーボードのような入力機器２２、磁気ディスクまたは光ディスクのような不揮発性記憶装置２３、プリンタ２４ならびに他の周辺機器というような周辺コンポーネントを有することができる。また、コンピュータプラットフォーム１２には典型的にはマイクロ命令コード２６およびオペレーティングシステム１２８も含まれている。

【００３４】図３に示されているように、オブジェクト指向コンピューティング環境１１はコンピュータプラットフォーム１２上で動作する。当業者であれば理解できるように、オブジェクト指向コンピューティング環境は多数のコンピュータプラットフォームにわたり動作可能である。オブジェクトコンピューティング環境１１は有利にはＣ＋＋コンピュータプログラミング言語で記述される。コンピュータプラットフォームのデザインとオペレーション、ならびにオブジェクトマネージャのデザインとオペレーションを含むオブジェクト指向コンピュー

ティング環境は当業者に周知であり、たとえばアメリカ合衆国特許第５，４９９，３６５号、１９９３年１１月２３日にAbraham 等に付与されたアメリカ合衆国特許第５，２６５，２０６号 " A Messenger and Objekct Manager to Implement an ObjectOriented Environment" 、Abraham 等に付与されたアメリカ合衆国特許第５，１６１，２２５号 " Presistent Stream for Processing Time Consuming and Reusable Queries in an Object Oriented Database Management System、Abraham 等に付与されたアメリカ合衆国特許第５，１５１，９８７号" Recovery Objectsin an Object Oriented Computing Environment" 、ならびに Abraham 等に付与されたアメリカ合衆国特許第５，１６１，２２３号 " Resumeable Batch Query for Processing Time Consuming Queris in an Object Oriented Database Management System" 、さらには数多くの教科書たとえば Bertrand Meyer 著、１９８８年 Prentice Hall 刊の " Object Oriented Software Construction" に示されている。これらは本出願の参考文献として挙げられるものである。

【００３５】さて、次に図１を参照すると、そこにはアメリカ合衆国特許第５，３１３，６２９号の図５から転載された図面が示されており、この図にはオブジェクト指向プログラムのメインコンポーネントが示されている。オブジェクト指向プログラムのデザインとオペレーションの詳細については、 Bertrand Meyer 著、１９８８年 Prentice Hall 刊の " Object Oriented Software Construction" で述べられている。

【００３６】図１に示されているように、典型的なオブジェクト指向コンピューティング環境１１には３つの基本的なコンポーネントが含まれている。すなわち、メッセンジャ５１とオブジェクトマネージメントテーブル５２とローデッドクラステーブル５３である。メッセンジャ５１は、コールする側とコールされる側のメッセージ、オブジェクトマネージメントテーブル５２、ローデッドクラステーブル５３の間の通信をコントロールする。オブジェクトマネージメントテーブル５２には、すべてのアクティブなオブジェクトインスタンスへのポインタのリストが格納されている。ローデッドクラステーブル５３には、アクティブなオブジェクトクラスのすべてのメソッドへのポインタのリストが格納されている。

【００３７】次に、図１に示された実例に関してオブジェクト指向プログラム１１のオペレーションについて説明する。この場合、あるオブジェクトのメソッドＡ（ブロック５４）が別のオブジェクトのメソッドＢ（ブロック５５）へメッセージを送信する。メソッドＡは、メッセンジャをコールすることによりメソッドＢへメッセージを送信する。このメッセージには１．メッセージを受信するためのインスタンスのオブジェクトリファレンスと、２．データにおいてカプセル化を実行するために必要とされるメソッドすなわちオブジェクトインスタンスと、３．メソッドの受信により必要とされるパラメータとが含まれている。メッセンジャ５１は、インスタンスオブジェクトに関してオブジェクトマネージメントテーブル５２をサーチすることで、メソッドＡにより指定されたインスタンスオブジェクトのデータフレーム５６へのポインタを獲得する。指定されたインスタンスオブジェクトを見つけられなければ、オブジェクトマネージメントテーブル５２はテーブルにそのインスタンスオブジェクトを追加し、データベースからのそのデータをマテリアライズするためにインスタンスをコールする。すでにインスタンステーブル中にあれば、オブジェクトマネージメントテーブル５２はマテリアライズされたインスタンスオブジェクトへのポインタを返す。

【００３８】次に、メッセンジャ５１はローデッドクラステーブル５３からメソッドＢのアドレスを獲得する。インスタンスのクラスがロードされていなければ、そのデータをマテリアライズするためこの時点でローデッドクラステーブル５３はそれをロードすることになる。ローデッドクラステーブル５３には指定されたメソッド（メソッドＢ）をサーチし、メッセンジャ５１へメソッドのアドレスを返す。

【００３９】次にメッセンジャ５１はメソッドＢをコールし、それへシステムデータエリアと、ポインタを含むメソッドＡによりなされたコールからのパラメータを渡す。メソッドＢはそのポインタを用いてデータフレーム５６をアクセスする。そしてメソッドＢはメッセンジャ５１へ制御を戻し、メッセンジャ５１はメソッドＡへ制御を戻す。

【００４０】図２にはオブジェクト指向コンピューティングプラットフォームにおける継承ハイアラーキの実例が示されている。図示されているように、" 販売員" 、" 従業員" および" 人物" という３つのオブジェクトクラスが描かれており、この場合、販売員は従業員の" 一種" であり、従業員は人物の" 一種" である。換言すれば、販売員は従業員のサブクラスであり、従業員は販売員のスーパークラスである。同様に、従業員は人物のサブクラスであり、人物は従業員のスーパークラスである。図示されている各クラスは３つのインスタンスを有している。この場合、B. Soutter, W. Tipp および B.G.Blue は販売員である。B. Abraham, K.Yates および R. Moore は従業員である。また、J. McEnroe, R. Nader および R.Reagan は人物である。換言すれば、１つのインスタンスは" ～は" という関係によってそのクラスに関連づけられている。

【００４１】各サブクラスは、そのスーパークラスのフレームおよびメソッドを" 継承する" 。したがってたとえば、販売員フレームは従業員のスーパークラスから年齢と給与データオブジェクトも継承するし、従業員スーパークラスから昇給メソッドも継承する。販売員はただ１つの割当て属性と賃金歩合メソッドも有している。各

インスタンスはそのスーパークラスのすべてのメソッドとフレームにアクセスできるので、たとえばB.G. Blueを昇進させることができる。

【００４２】先に述べたように、本発明のＳＥＡＳＭは、コンカレンシイ、ディスパッチングおよび同期化のためのアプリケーションプログラマインタフェースが提供されるように設計されたものであり、これによりアプリケーションプログラマはローレベルのスレッドの詳細に煩わされることから解放される。このように、ＳＥＳＡＭは同期および非同期のファンクション動作に対するゲートとして用いられる。

【００４３】次に図４を参照するとそこに示されているように、１つのオペレーティングアクセス１００内においてアプリケーションレベル１０２ではたらくアプリケーションプログラマは、プロセスのスタートアップに基づきメインディスパッチャ１０４にサービスを登録する。しかし、登録されたサービス内で非同期が必要とされるときにはいかなる場合でも、そのサービスはＳＥＳＡＭ１０６を利用することができる。その場合、ＳＥＳＡＭディスパッチャ１０８により、非同期コールバックスケージューリングのサービスが行われる。これには非同期で実行されるよう登録されたすべてのユーザコールバックが含まれている。

【００４４】この場合、メインディスパッチャ１０４とＳＥＳＡＭディスパッチャ１０８の両方は、メインディスパッチャ１０４をブロックすることのできるユーザコールバックコードを実行することになるので、第３のディスパッチャであるＳＥＳＡＭシグナリングディスパッチャ１１０が含まれており、これはタイマスケージューリングと通信イベントがユーザコードにより遅延されないようにするためのものである。ユーザコールバックはＳＥＳＡＭに登録され、これは適正なディスパッチャに対しコールバックをスケジューリングするよう通知する。システムの例外はメインスレッド内でローレベルにおいて処理され、このメインスレッドの方はディスパッチャまたはスレッドに対し例外コールバックを実行するよう通知する。

【００４５】これまでのことを理解できるようにするために、次にＳＥＳＡＭの機能について説明する。

【００４６】ヒューリスティックス

エベントは常に非同期に発生する。プログラムのセマンティクスに依存して、（登録されたコールバックを介して）非同期でまたは（イベントが発生したか否かに関して明示的なクエリーにより）同期して通知を行うのが望ましい。その他の重要な目的は、非同期に到来するイベントに対する同期化である。つまりプログラマはいくつかの処理を望んでおり、所定の時点においてプログラマは次のイベントまたはイベントの論理的な組み合わせが生じるまでブロックを行いたいと望む。

【００４７】機能

ＳＥＳＡＭは、ＡＳＹＮＣ－ＡＳＹＮＣプログラミングモデルもＡＳＹＮＣ－ＳＹＮＣプログラミングモデルもサポートしようというものである。登録されたコンシューマに対し非同期のイベントを非同期にプロモートすべきであるならば（プッシュモード、サプライア（生産者）トリガ型）、ＡＳＹＮＣ－ＡＳＹＮＣプログラミングモデルが有利である。ＡＳＹＮＣ－ＳＹＮＣプログラミングモデルは、イベントをいつトリガすべきであるのかをコンシューマ（消費者）が決めるときに有利である（コンシューマドリブン型、プルモード）。

【００４８】ＳＥＳＡＭは、以下の機能に対する同一性の快適なインタフェースが提供されるように設計される。

【００４９】・非同期での機能の実行（つまり別個のスレッドにおける機能の実行）（非同期

・ファンクションマネージャ＝ＡＦＭ）

・同期タイマのスケジューリング

・非同期タイマのスケジューリング

・例外の処理

・通信イベントのための汎用的な同期化インタフェースを提供

・イベントリストを待つ

・多数のディスパッチャの同期化

上述のサービスの各々は"スロット"を内部的に利用する。ＳＥＳＡＭにおけるスロットは管理上のエンティティであり、これはスロットに割り当てられたアクティビティの状態を追従する。これらのスロットは（ユーザには見えない）特別な目的に合わせて作られており、ユーザに見えるSynchHandlesと呼ばれる同期化ハンドルにより定義される。この SynchHandles により、ＳＥＳＡＭにより供給されるサービスに対し同種のビューが提供される。SynchHandlesのライフサイクルは、ＳＥＳＡＭオブジェクトのライフサイクルに縛られていく。

【００５０】デバッグおよび／またはサポートの目的で、各スロットに（唯一であることをチェックせずに）１つの名前を与えることができる。つまり、ランタイムにスロットに関して重要な情報を取り出すことができる（たとえば型、名前、状態等）。

【００５１】呼び出しのためにＳＥＳＡＭに従属する関数を、特別なスレッド固有のデータセッティングに依存させることができる。しかし、ＳＥＳＡＭによれば内部的な関数呼び出しのための（スレッドのための）実行エージェントが設けられているので、アプリケーションプログラマは関数呼び出し前と呼び出し後にスレッド固有のデータをセットしセーブすることができる。

【００５２】フックは、HookInfo 構造体を利用する選択されたＳＥＳＡＭ ＡＰＩ 関数の呼び出しに応じてアプリケーションプログラムにより指定することができる。この場合、HookInofo 構造体により、関数の各々について４つの関数をパラメータとともに規定することが

できる。

【００５３】図５には、ＳＥＳＡＭの基本コンポーネントが示されている。この場合、Ｃ＋＋のクラスのヘッダファイル CsaSESAM により、ここで述べるコンポーネントがインプリメントされる。とりわけクラス CsaConnectable および Csallemote のようなＳＥＳＡＭにより利用される他のアイテムに関するその他の情報は、たとえばアメリカ合衆国特許出願 Serial No._，Attorney Docket No. P96,0462 参照。

【００５４】図示されているように、ＳＥＳＡＭには５つの異なるタイプのスロットが含まれており、各々のスロットは以下の目的に合わせて作られている：

－アプリケーションプログラマのための非同期を処理するダイナミックスロット２００

－同期タイマを処理するタイマスロット２０２（同期）

－非同期タイマを処理するタイマスロット２０４（非同期）

－ユーザ定義システム例外コールバックを処理する例外スロット２０６

－ユーザイベントつまり外部のイベント通知のための waitFor...() 機能の快適性を提供する汎用スロット

スロットは結局はスレッドを利用する必要があり、ユーザにより選択された呼び出しモードに従ってコールバックをスケジューリングする。したがって、スレッドマネージャ２１０によりスレッドファクトリが形成され、上述のような複数のディスパッチャ１０４，１０８，１１０は適切なスケジューリングによりコールバックを呼び出すためのものである。図示されているようにメインディスパッチャ１０４に関して、メインディスパッチャ１０４自体はＳＥＳＡＭの一部分ではないので、ＳＥＳＡＭはそれに対するポインタ１０４Ａを設けている。

【００５５】Synchllandles は、割り当てられているスロットタイプにかかわらずＳＥＳＡＭ内のアクティビティを定義する。ＳＥＳＡＭによりサービスされるすべてのアクティビティは Synchllandles を通して参照されるので、異なるスロットおよびアクティビティに対する均質なビューが提供される。内部的に、各スロットは生成時にシーケンシャルにナンバリングされる。殊に、各スロットは再利用可能であるので（つまり、それらが１つのアクティビティを終了させてしまったとき、それらは次のリクエストの処理を開始することができるので）、目下の Synchllandle とこのアクティビティを処理する内部的なスロット識別子との間のマッピングが必要である。これはスロットアドミニストレータ２１２の役割である。

【００５６】メッセージブロックのプールにより、効率のよい内部的なメッセージ伝達メカニズムを使用できるようになる。この場合、メッセージブロックも再利用可能である。利用可能なメッセージブロックは、メッセージブロック蓄積部２１４内に保持される。

【００５７】たとえば除去されたあるいはうまく完了しなかったアクティビティから waitFor...() インタフェースにおいて Synchllandles を用いた場合、結果としてエラーになる。したがって、このようなエラーのあるアクティビティに対して割り当てられていたスロットを再利用できるようにする一方で、そのような Synchllandle に関する履歴を失わないようにする目的で、＂無効な＂Synchllandles のリスト２１６がＳＥＳＡＭ内に格納されている。

【００５８】さらに、waitFor_...() コール内で目下ブロックされているスレッドのためのアドミニストレータ２１８が設けられている。このアドミニストレータ２１８は、あるスレッドが待っている条件が真になったとき、そのスレッドをブロックしないようにする。

【００５９】ユーザ関数（ファンクション）の非同期の実行
(ASYNCHRONOUS EXECUTION OF USER FUNCTIONS: AFM) ＳＥＳＡＭにより、アプリケーションプログラマに対するコンカレンシイ・メカニズムが提供される。これによりプログラマは同期にパラレルに実行できるようになり、つまりそれらの関数は（たとえば複数の処理ユニットのパラレルな使用を効率的にする目的で）互いに対して非同期に実行される。

【００６０】非同期をインプリメントするため、ＳＥＳＡＭ内でダイナミックスロットが使用される。

【００６１】ユーザ固有の関数は、その関数の実行を依頼したスレッドに対して非同期に１つのスレッドにおいて呼び出される。この場合、実行状態が自動的に監視され、それに対しいつでもクエリーを行うことができる。その際、非同期に実行される関数の終了に応じた同期化も行われるし、非同期の関数の終了に応じたユーザ定義コールバックのアクティブ化も行われる。このコールバックはメインディスパッチャ１０４から同期して呼び出すこともできるし、あるいはＳＥＳＡＭディスパッチャ１０８または別個のスレッドから非同期に呼び出すこともできる。

【００６２】ユーザ定義関数へ引数を渡すことができる。ユーザ定義関数の戻り値は、引数としてコールバックへ自動的に渡される。コールバックは値を返すことはできない。

【００６３】図６には、ユーザ関数を非同期に実行するための基本原理が示されている。

【００６４】図示されているように、メインディスパッチャからユーザへ渡される非同期の関数は、別個のスレッドにおいて実行させるようユーザコールバックリクエストにより与えられ、つまり同期関数はメインスレッドに対し非同期に実行されることになる。

【００６５】同期関数の非同期の実行が完了すると、ユーザ指定コールバック関数が呼び出される。同期関数の戻り値は引数としてコールバックへ渡される。

【００６６】コールバックを実行すべきコントロールのスレッドは、呼び出しよりも前に選択する必要がある。このようなコントロールの選択の種類としては、
－メインディスパッチャスレッド
－ＳＥＳＡＭディスパッチャスレッド
－同期コールをすでに実行した非同期スレッドが挙げられる。

【００６７】コールバックメカニズムは非同期に実行された関数の戻り値を処理するのにリーズナブルな手法であると思われるが、後になって（コントロールの流れと同期して）結果を処理するのがリーズナブルである場合もある。たとえば、ユーザがコールバックを指定しなかった場合、非同期に実行された関数の戻り値を、ユーザによる明示的なクエリーが行われるまで内部で格納しておくことができる。

【００６８】ダイナミックスロットの状態ならびにトランザクションダイナミックスロットの状態に対するクエリーを行う場合、以下のリストに掲載した状態を戻すことができる。

【００６９】Ｉｎｉｔｉａｌ－
シグナリングなし、スロットは初期状態にあり未使用。

【００７０】Ｓｔａｒｔｅｄ－
シグナリングなし、スロットは使用中、スレッドはユーザ関数の実行を開始。

【００７１】Finished-
シグナリングあり、スロットは使用中、ユーザ関数は終了。

【００７２】CbActive-
シグナリングあり、スロットは使用中、コールバック関数実行中。

【００７３】Idle-
シグナリングあり、スロットは未使用、再利用可能。

【００７４】図７には、１つのダイナミックスロットに関する状態遷移が示されている。図７の場合、丸で囲んだ数字は次の括弧内の参照符号で表された状態遷移に対応するものである。

【００７５】図示されているように、生成と同時にスロットは”Initial”（初期）状態３００となる（これは以前は未使用であったことを表しそのメンバすべては無効な値を有する）。１つの関数が弓数およびコールバックとともにＳＥＳＡＭに対し”activate()”インタフェースを介して実行依頼されると、スロットは”Queued”（キューイング）状態３０２に入り、つまり状態遷移（１）を経る。ワーカスレッドが生成されユーザ関数の実行が開始されると、スロットは”Started”（開始）状態をとり、つまり状態遷移（２）を経る。ユーザ関数が戻ると、スロットは”Finished”（終了）状態３０６をとり、つまり状態遷移（３）を経る。

【００７６】その後、コールバックがアクティブにされると、スロット状態は”CbActive”（コールバックアクティブ）状態３０８に変わり、つまり状態遷移（４）を経る。コールバックが戻るとただちに、スロットは”Idle”（アイドル）状態３１０となり、つまり状態遷移（５）を経る。新たなアクティブ化とともにスロットは再び”Queued”（キューイング）状態３０２に入り、つまり状態遷移（１）を受ける。

【００７７】”Finished”状態３０６から”Idle”状態３１０への図示の状態遷移（６）は、ユーザがコールバック関数を指定せず、ユーザ関数の戻り値がダイナミックスロット中に保持されるよう指定した場合にトリガされる。この場合、スロットは、このダイナミックスロットに対する戻り値がユーザにより明示的にクエリーされるまで”Finished”状態３０６のままである。最初のクエリー後、スロット状態は”Idle”状態３１０へ変化する。ユーザがコールバックを指定せず、かつユーザ関数の戻り値の保持も望んでいなければ、状態遷移（６）はスロットが”Finished”状態３０６になるとただちに自動的に実行され得る。

【００７８】図８には、コールバックモード CB_ASYNC_MD_ASYNC_AD により非同期に１つの関数の実行される様子が示されている。

【００７９】丸で囲んだ数字は、次節中の括弧内の数字と、ＳＥＳＡＭの状態レジスタの状態に関する状態遷移ダイアグラム中の丸数字とに対応するものである。

【００８０】図８に示されているように、スレッドＡがＳＥＳＡＭに対し１つの関数および１つのコールバックを実行依頼した場合、つまり遷移（１，７）を生じさせようとした場合、ＳＥＳＡＭはこの要求を満たすため内部的に１つのスロットを必要とする。ダイナミックスロットが存在していないかまたは利用できなければ、暗黙的に新たなダイナミックスロットが生成される。１つの関数が１つのダイナミックスロットへ渡されると（ユーザには隠蔽された）新たなスレッドが生成され、このスレッドは実行依頼された関数の実行を開始させ、つまり遷移（２）を経ることになる。ユーザ関数が戻ると、スロット状態は自動的に”Finished”状態に変わり、つまり遷移（３）を経る。コールバックが指定されていたならば、次にそれが実行され、つまり遷移（４）が生じる。このコールバックは（図示のように）同じスレッドにおいて実行させることができ、あるいは（図示されていない）ディスパッチャスレッドにおいて次に実行させるためディスパッチャによりスケジューリングされる。コールバックが終わると、スロット状態は再び”Idle”状態に変わり、つまり遷移（５）が行われる。あらゆるスロット状態変化は暗黙的に（ユーザには隠蔽されて）行われる。１つの関数の実行依頼により、ダイナミックスロットはシグナリングなし状態となる。ユーザ関数の実行終了時点で、ダイナミックスロットは自動的にシグナリングあり状態にセットされる（これによりそのスロットがシグナリングあり状態になるのを待っていたスレ

ッドたとえばスレッドAがリリースされる）。

【００８１】コールバック呼び出しモード

コールバックが指定されている場合、３つのコールバック呼び出しモードのうちの１つを選択する必要がある。以下の表１にそれらのモードを示す。

【００８２】

【表１】

	CbSyncMainDispatch	CbAsyncMainDispatch_-SyncSesamDispatch	CbAsyncMainDispatch_-AsyncSesamDispatch
ダイナミック動作	メインディスパチャにより同期してコールバックが呼び出される。	内部的なＳＥＳＡＭディスパチャによりコールバックが呼び出される。これはメインディスパチャに対し非同期であり、つまりメインディスパチャに登録されたコールバックに対しパラレルにコールバックが呼び出される。	ユーザ関数と同じスレッドでコールバックが呼び出される。
特別な機能	通知が戦闘的にクリティカルでないパラレルに他のメインディスパッチングが実行されないときに使用される（たとえばコールバックがスレッドセーフでないとき）。	このモードに登録されたすべてのコールバックは同じディスパッチャにより呼び出されるので、それらはパラレルには実行できずシリアルに実行される（たとえばコールバックがスレッドセーフであればこれが有用）。しかしメインディスパッチャに対する並列性はある。	関数の終了とコールバックの呼び出しの間が最小遅延のもっとも精確な通知メカニズム、つまり最大の並列性が得られる。
特記	長時間にわたり他のコールバックにより遅延される可能性あり。コールバックがブロッキングコールを使用すればメインディスパッチャにより呼び出される他のコールバックの実行が遅延される。	このモードに登録された他のコールバックにより遅延される可能性あり。またはブロッキングコードを用いればこのモードによる他のコールバックが遅延される。	コンカレンシイ状態が発生し得る。

表１：コールバック呼び出しモード

【００８３】コントロールインタフェース

ダイナミックスロットは一時停止、再開ならびに除去が可能である。この場合、実行に与える影響は、コールバック呼び出しモードと命令が発行される時点とに依存する。

【００８４】ダイナミックスロットは以下のときに一時停止される：

・コールバックモード CbSyncMainDispatch または CbAsyncMainDispatch_SyncSesamDispatch が呼び出されたとき、および

－非同期のスレッドが存在している間、一時停止によりスレッド一時停止が行われることになるとき。

【００８５】－スレッドがもはや存在していなければ、ダイナミックスロットに一時停止フラグがセットされる。ディスパッチャにより実行されるようコールバックがスケジューリングされ、一時停止フラグがセットされていれば、コールバック呼び出しがスキップされ、つまり通知が失われる。以後、ダイナミックスロットは通知が行われたかのように自動的に除去される。コールバックがすでにアクティブであれば、一時停止フラグの設定は無視される。

【００８６】－コールバックがすでに完了していれば、個々のダイナミックスロットはもはや存在せず、一時停止によりエラーが戻される。

【００８７】・コールバックモード CbAsyncMainDispatch_AsyncSesamDispatch が呼び出されたとき。この場合、

－ユーザ関数ならびにコールバック関数の全実行時間にわたりスレッドが存在しているので、スレッドはすでに一時停止されていることになり、その結果、ユーザ関数またはコールバックが一時停止される。この場合、通知が失われることはない。

【００８８】・KeepRetVal または DismissRetVal が呼び出されたとき、および

－非同期のスレッドが存在している間、一時停止によりスレッド一時停止が行われるようになるとき。モード DismissRetVal が指定されていた場合、スロットはスレッドが存在しているよりも長く存在せず、したがって一時停止に対しスロットがエラーを戻すよう試みる。モード KeepRetVal が指定されていて、スレッドがもはや存在していない場合、（まだ存在していれば）ダイナミックスロットに一時停止フラグがセットされる。一時停止

されたスロットの戻り値を取り戻すようプログラマが試みようとしたとき、プログラマはエラーメッセージを受け取ることになる。戻り値の取り戻しは、スロットが一時停止されていないときにのみ可能である。

【００８９】ダイナミックスロットは以下のときに再開される：

・スレッドが一時停止されていたとき。

【００９０】・スロットがまだ在住していて、一時停止フラグがリセットされるとき。

【００９１】・スロットが存在しておらず、再開によりエラーコードが戻されるようになるとき。

【００９２】ダイナミックスロットは以下のときに除去される：

・コールバックモード CbSyncMainDispatch または CbAsyncMainDispatch_SyncSesamDispatch が呼び出されたとき。

【００９３】－非同期のスレッドが存在しているかぎり、スレッドが切られることになり、相応のイベントハンドラをディスパッチャから除去する必要がある。

【００９４】－スレッドがもはや存在していなければ、ダイナミックスロットに除去フラグがセットされる。ディスパッチャにより実行されるようコールバックがスケジューリングされており、除去フラグがセットされていれば、コールバック呼び出しはスキップされ、つまり通知は失われる。移行、ダイナミックスロットは、通知がなされたかのように自動的に除去される。コールバックがすでにアクティブであれば、除去フラグの設定は無視される。

【００９５】－コールバックがすでに完了していれば、相応のダイナミックスロットはもはや存在せず、除去によりエラーが戻される。

【００９６】・コールバックモード CbAsyncMainDispatch_AsyncSesamDispatch：－この場合、ユーザ関数およびコールバックの全実行時間にわたりスレッドが存在しているので、スレッドはすでに切られていることになり、その結果、ユーザ関数のキルまたはコールバックのキルが行われる。

【００９７】・KeepRetVal，DismissRetVal：

－非同期スレッドが存在している間、除去によりスレッドのキルが行われることになる。

【００９８】－スレッドが存在しておらずスロットがまだ存在していれば、除去によりスロットが自由になる。スロットがもはや存在していなければ、除去によりエラーが戻される。

【００９９】除去により、セマンティクスに留意することなくいかなる時点でもアクションが中断されることになり、つまり不完全なトランザクションが生じる可能性がある。

【０１００】シグナリングあり状態

ダイナミックスロットは、ユーザ関数が非同期スレッド内で戻されたときにはいつもシグナリングあり状態となる。シグナリングあり状態は、あとでスロットが一時停止／除去されてもリセットできないし変更されない。waitFor...() コール内で非同期関数が完了する前に除去されたダイナミックスロットの SyncHandle を使用すると、エラーが生じることになる。waitFor...() コール内で非同期関数が完了した後に除去されたダイナミックスロットの SyncHandle を使用すれば、正常にはたらくことになる。いかなる論理的な組み合わせにおいても非同期関数の完了を待っているどのような waitFor...() も、相応のダイナミックスロットが除去された時点で中断され、エラーを戻すことになる。

【０１０１】" activate()" インタフェースにより、ユーザ関数" beforeFunc" が呼び出される前、ユーザ関数" afterFunc" が戻された後、コールバック" beforeCb" がアクティブにされる前、ならびにコールバック" afterCb" が戻された後、呼び出すべきフックの指定が可能になる。関数" beforeCb" と" afterCb" は、コールバックが指定されていた場合にのみ呼び出される。

【０１０２】また、関数" beforeFunc" および" afterFunc" は、その関数自体と同じスレッド内で呼び出される。関数" beforeCb" と" afterCb" は、コールバック関数自体を実行するスレッドのコンテキストで呼び出される。

【０１０３】スケジューリングタイマ

指定された時間遅延後にアクティビティをスタートさせるためにタイマが用いられる。そのようなアクティビティはたいてい、ユーザ指定関数の呼び出しであることが多い。ユーザ関数は、タイマが終了してから実行されるようスケジューリングされる。タイマスロット２０２と２０４（図５）は、タイマコントロールされるアクティビティをサポートするためＳＥＳＡＭ内で使用される。

【０１０４】ＳＥＳＡＭは、タイマをスケジューリングする（つまりユーザ関数をスケジューリングする）２つのモードすなわち同期および非同期をサポートしている。

【０１０５】タイマの同期したスケジューリングに関していえば、タイマが終了した後、できるだけはやくユーザ関数を実行させるよう、メインディスパッチャ１０４に通知される。これは、ディスパッチャ１０４がまだ別のユーザ関数を実行している場合には、しばらくの間かかる可能性がある。複数のタイマが同時に完了するようスケジューリングされていた場合であると、登録されたユーザ関数はシーケンシャルに実行され、その結果、関数を実行させようとした本来の所望の時点に対し遅延が増大する。

【０１０６】タイマの非同期のスケジュールに関していえば、タイマが終了した後、できるだけはやくユーザ関数を実行させるよう、ＳＥＳＡＭディスパッチャ１０８に通知される。これは、ディスパッチャ１０８が別の非

同期に登録されたコールバック関数をまだ実行しているのであれば、しばらくの間かかる可能性がある。複数のタイマが同時に終了するようスケジューリングされていた場合であると、登録されたユーザ関数はシーケンシャルに実行され、その結果、関数を実行させようとした本来の所望の時点に対し遅延が増大する。しかし、非同期のタイマのコールバックは、メインスレッドで実行される関数に対し非同期で実行されることになる。

【０１０７】図９には、ワンショットタイマに関するアクティビティのシーケンスが示されている。ここには以下のことが示されている：

（１）タイマは終了すべき時点が与えられることにより具象化され、タイマの終了に応じてコールバック関数が呼び出される。

【０１０８】（２）その後、タイマが終了したときに、コールバックを呼び出すよう指定されたディスパッチャに通知が行われる。

【０１０９】（３）そのディスパッチャによりコールバックが呼び出される。タイマの終了（２）とコールバックの呼び出し（３）との間の遅延は、ディスパッチャが別のコールバックの呼び出しにどの程度かかわっているかに依存する。

【０１１０】インターバルタイマの指定にあたり２つの値を与える必要がある。すなわち、

・遅延値：この時間値は起動してから最初にタイマが終了する時間である。

【０１１１】・インターバル値：この時間値は順次連続する２つのタイマ終了の間の△期間である。

【０１１２】図１０には、インターバルタイマに関するアクティビティのシーケンスが示されている。この図に示されているように、相応のコールバックが呼び出される前にインターバルタイマが終了すると、終了イベントが内部的にキューイングされる。

【０１１３】図１１には、時点（Ａ）においてトリガされたコールバックは、ディスパッチャがビジーであるため呼び出せない様子が示されている。コールバックが呼び出される前、タイマは第２の時点で終了し、その後でしかディスパッチャはコールバックを開始させることができない。しかしタイマ終了および呼び出されるコールバックの数はやはり同じである。

【０１１４】コントロールインタフェース

ＳＥＳＡＭの場合、タイマスロットの一時停止、再開および除去が可能である。タイマスロットの一時停止により、その時点でコールバックの呼び出しが抑圧される。タイマスロットの一時停止によっても、そのシグナリングあり状態の動作は変わらない。タイマスロットの再開により、コールバックの呼び出しが再び可能になる。タイマスロットの除去により、ＳＥＳＡＭ内のタイマスロットが削除される。タイマスロットが除去されると、その SynchHandle は、この場合にはエラーを戻すことになる wailFor...() に対して無効になる。

【０１１５】図１２および図１３には、タイマスロットの一時停止の作用が示されている。図１２には、コールバックの先行する抑圧の様子が示されている。図１３には、インターバルタイマスロットのシグナリング状態が示されている。

【０１１６】シグナリングあり状態

タイマがワンショットタイマであれば、そのスロットは（タイマがその時点で一時停止されているか否かにかかわらず）タイマが終了した後、シグナリングあり状態となり、その状態はリセットできない。タイマがインターバルタイマであれば、シグナリングあり状態は（タイマがその時点で一時停止されているか否かにかかわらず）各終了時間においてのみパルス化される。タイマが除かれればそのスロット状態は無効になり、つまり wailFor...() コールにおいてもはや利用できない。いかなる論理的な組み合わせにおいてタイマイベントを待つどのような wailFor...() コールも中断され、関係するタイマスロットが除去された時点でエラーが戻される。

【０１１７】" addTimer()" インタフェースにより、タイマコールバックがアクティブにされる前（Cb コンポーネントの前）およびタイマコールバックが戻された後（Cbコンポーネントの後）、呼び出すべきフックの指定が可能になる。それらのフック関数は、タイマコールバックが呼び出される同じスレッドにおいて呼び出される。

【０１１８】システム例外の処理

あるプロセスが完全にユーザレベルでハングしている場合（たとえば無限ループでハングアップしていれば）、そのプロセスが無視できない高い優先度のメッセージをそのプロセスへ発行するのが望ましい。このように高い優先度のメッセージを例外と呼ぶ。ＵＮＩＸシステムの場合、例外は信号としてインプリメントされている。この場合、プログラマは、例外が発生したときに呼び出されるべきそのような例外に対するコールバックを登録しておき、例外に対して応答させるようにすることができる。このようなサービスをインプリメントするため、ＳＥＳＡＭでは例外スロットが用いられる。

【０１１９】この目的で、システム例外（信号）を常に受け取るようにメインスレッドが構成される。デフォルトの例外ハンドラは例外の応答に応じてスタートする。このデフォルトハンドラは、選択された呼び出しモードに従って登録されたすべてのコールバックを呼び出すためのものである。この場合、コールバックのためにサポートされている呼び出しモードは、ダイナミックスロットのためのものと同じである（つまり CBSyncMainDispatch, CbAsyncMainDispatch_SynchSesamDispatch, CbAsyncMainDispatch_-AsyncSesamDispatch ）。

【０１２０】サポートされる各々の例外については、周知のただ１つの SynchHandle がある。この場合、例外

に対し（ダイナミックスロットに対するコールバックモードのように）異なる呼び出しモードでコールバックを登録することもできるし、あるいは次の例外を明示的に待つこともできる。なお、通知メカニズムもダイナミックスロットに用いられるものと同じである。ＵＮＩＸにおけるシステム例外は信号を利用してインプリメントされているので、以下のいくつかのことを考慮する必要がある：

・物理的に、システムは例外ごとに１つの例外ハンドラしか知らない（ハンドラのスタックではない）。ＳＥＳＡＭは各呼び出しモードごとに例外ハンドラの別個のスタックを提供する。しかし、サードパーティのライブラリにより、先にインストールされていたＳＥＳＡＭ例外ハンドラについて煩みないシステムとともに独自の例外ハンドラがインストールされる可能性がある。そしてこれにより、ＳＥＳＡＭの登録されたコールバックのための例外が失われる可能性がある。このような非協調的な動作を避けるため、ＳＥＳＡＭにより再インストール手法が提供されており、これによればサードパーティのハンドラを呼び出しに応じてその独自のコールバックルーチンとともに変更できるようになる。

【０１２１】・シグナルハンドラのインストール時に、例外信号により割り込まれるシステムコールにより何が起こるかを定義する必要がある。つまり、エラーを戻すべきか（割り込みついでユーザに通知するか）または内部的にリスタートすべきかを定義する必要がある。サードパーティライブラリのハンドラのコンフリクトを起こすセッティングは解決できない。つまり、あるライブラリがシステムコールの割り込みに依存しており他のものが（同じプロセス内で）そうではないとしたら、前者のライブラリコンポーネントはハングする可能性があり（これは後者のライブラリにより望まれるようにリスタートが選択されても前者のライブラリコンポーネントはシステムコールをそのままにして無視してしまうことがない理由による）、あるいは後者のライブラリがエラーを戻すことになる（これは前者のライブラリにより望まれるようにリスタートなしが選択されていても、後者のライブラリはシステムコールが割り込まれたことを予期しない理由による）。

【０１２２】・サードパーティのライブラリはランタイム時にシグナルハンドラをアンインストールしない可能性がある。たいていの場合にはおそらく、それらはそのデフォルトのシグナルハンドラをアンインストールすることになる。

【０１２３】デフォルトのハンドラのインストール中に見つけられるサードパーティの例外ハンドラは、デフォルトのハンドラによりシグナルハンドラレベルに基づき呼び出される。ＳＥＳＡＭに登録された例外ハンドラはシグナルハンドラレベルに基づき呼び出されるのではなく、選択された呼び出しモードにおけるユーザレベルに基づき呼び出される。各々の例外についてＳＥＳＡＭ内には３つまでのタスクが存在し得る：

・CbSyncMainDispatch により登録されたコールバックを配慮するタスク

・CbAsyncMainDispatch_SyncSesamDispatch により登録されたコールバックを配慮するタスク

・CbAsyncMainDispatch_AsyncSesamDispatch により登録されたコールバックを呼び出すタスク

この結果、１つのタスク内で同一の呼び出しモードのためのコールバックのスタックが保持され、各コールバックがシーケンシャルにアクティブにされる。デフォルトのハンドラは、１つのメッセージを各タスクのためのメッセージキューに入れるだけである。

【０１２４】図１４には、（１つのスロット内でつまり１つの例外のために）呼び出しモードにより分けられたユーザ定義システム例外ハンドラのスタックが示されている。図示されているように、異なる呼び出しモードにおいて同じ例外のために登録された例外ハンドラはパラレルに実行できる。それというのも、それらは異なるスタックに配置されているからである。

【０１２５】図１５には、同じ例外のための例外ハンドラがパラレルに実行される様子が示されている。図示されているようにこのような設計によれば、コールバックモード“CbAsyncMainDispatch_AsyncSesamDispatch”を用いて登録されていれば、異なる例外についてパラレルに例外ハンドラを実行させることも考慮される。他のすべての呼び出しモードにより、（たとえ異なる例外に対してでも）コールバックのシーケンシャルな実行が行われる。

【０１２６】図１６には、（コールバックモード CbSyncMainDispatch を用いる）ディスパッチャ１０４または（コールバックモード CbSyncMainDispatch_SyncSesamDispatch を用いる）ディスパッチャ１０８を介して呼び出すことで行われる例外ハンドルーチンのシリアル化の様子が示されている。

【０１２７】図１７には、コールバックモード CbAsyncMainDispatch_AsyncSesamDispatchを用いた場合に、異なる例外のための例外ハンドラがパラレルに実行される様子が示されている。

【０１２８】デフォルトのハンドラの各々の呼び出しによって、関連する例外のためのスロットがパルス化され、つまりセットおよびリセットされる。

【０１２９】シグナリングあり状態

図１８には、１つの例外スロットのシグナリングあり状態が示されている。図示されているように、例外スロットのための Sync ハンドルは常に有効であり、つまり wailFor...() コールにおいて常に利用できる。

【０１３０】信号レベルに基づき最初に呼び出されたとき、あるいはサードパーティのライブラリからの協調的なシグナルハンドラにより繰り返し呼び出されたとき、

デフォルトのハンドラは検出の役割を担う。繰り返し呼び出されたとき（つまり無限ループのとき）、デフォルトのハンドラはこのアクティビティを実行することなくそのままもとに戻る。

【０１３１】デフォルトのハンドラのインストールに応じて、すでにインストールされているサードパーティソフトウェアのハンドラを、登録されている他のすべてのコールバックといっしょに拘束しなければならない。

【０１３２】デフォルトのハンドラ"SA_RESTART"および"SA_NORESTART"のインストールモードは、インストール時に見つけ出されたサードパーティライブラリのハンドラのインストールモードに依存することになる。デフォルトハンドは常に、先に登録されたサードパーティライブラリのハンドラと同じモードでインストールされるようにする。サードパーティハンドラは、コンフリクトを起こすインストールモードをもっていてはならない。

【０１３３】サードパーティのライブラリは、ランタイムにシグナルハンドラをアンインストールしてはならない（その理由はそれらはたいていデフォルトのハンドラをアンインストールすることになるからである）。

【０１３４】特定の信号のためのデフォルトのシグナルハンドラのインストールは、最初のコールバックがこの信号に対して（register()により明示的にまたはwaitFor...()により暗黙的に）登録されるときにのみ必要である。

【０１３５】各々の例外のためのタスク内でのメッセージキューの操作は、デッドロックを避ける目的でシグナルガードを利用する必要がある。

【０１３６】安全のため、シグナルハンドラレベルでアクティブな各スレッド内で、シグナルハンドラレベルでの実行を指示するためスレッド固有のストレージエントリがセットされる。このエントリにより、非例外のセーフファンクション内でそれらが適正に呼び出された否かについてチェックすることが配慮される。例外レベルで呼び出されたのであれば、それらはエラーを戻すことになる。このスレッド固有のストレージエントリのセットおよびリセットは、登録されたコールバックを順次呼び出すための環境を提供する関係するハンドル...()のルーチンの役割である。

【０１３７】デフォルトシグナルハンドラへのエントリに基づき、ユーザからのスレッド固有のエラーを（エラーの上書きを避けるため）セーブしておき、ハンドラを除去する前にリストアする必要がある（このことでユーザレベルでデータが再び得られるようになる）。

【０１３８】例外ハンドラは、"remove"コールが発行されたとき、たとえそのハンドラについて例外がまだペンディングしていても除去される。

【０１３９】"registerExceptionHandler()"インタフェースにより、例外コールバックがアクティブにされる前（Cb コンポーネントの前）および例外コールバックが戻された後（Cb コンポーネントの後）、フックの指定を呼び出すことができる。フック関数は、例外コールバックが呼び出されている同じスレッド内で呼び出されることになる。

【０１４０】汎用同期インタフェース

パラレルなアクティビティの完了に基づく同期化は重要なポイントである（たとえば SyncHandle 1 または SyncHandle 2 が完了するまで待機するが、これは一定の長さの時間よりも短い）。エラーを起こしやすいこのようなジョブにアプリケーションプログラマが煩わされないようにする目的で、ＳＥＳＡＭにおいて汎用スロットコンセプトが設けられており、これによればあらゆる種類のイベントに対し同種の waitFor...() を使用できるようになる。

【０１４１】ＣＳＡのようなソフトウェアアーキテクチャにおけるイベントプロパゲーションは、CsaConnectables や CsaRemotes のようなオブジェクトまたはクラスによりインプリメントされているので、それらは内部的にＳＥＳＡＭの汎用スロットインタフェースを利用し、これによってユーザはいかなるユーザイベントも待つことができるようになる（つまりどのようなユーザイベントにも同期することができる）。CsaConnectables および CsaRemotes は、基盤となる（下位層の）基本通信システム（basic communication system, "bcs" はユーザには隠蔽されている）と内部的につながっている。これらのオブジェクト／クラスについては、上述の特許出願を参照のこと。

【０１４２】汎用スロットは生成、除去、シグナリングあり状態へのセット、シグナリングあり状態からのリセット、およびパルス化（シグナリングあり状態の暗黙的なセット／リセット）が可能である。汎用スロットはそれに割り当てられた SyncHandle を有しているので、それを待つことができる。これにより、いかなるユーザイベントについても waitFor...() コールの適用範囲が拡げられる。

【０１４３】汎用スロットは、"CsaConnectable"（つまりユーザイベントのサプライアないし供給者）および bcs オブジェクトの組み合わせと相互接続されており、あるいは bcs オブジェクトと"CsaRemote"オブジェクト（つまりユーザイベントのコンシューマないし消費者）と相互接続されている。

【０１４４】サプライア接続

汎用スロットは bcs オブジェクトによって、それに CsaConnectable が登録されるたびに生成される。この汎用スロットは、もっぱら新たに登録された CsaConnectable に対してのみ割り当てられる。必要であれば、CsaConnectable はその bcs からの SynchHandle のクエリーを行うことができる。

【０１４５】図１９には、CsaConnctable と汎用スロッ

トと bcs オブジェクトとの間の相互接続について示されている。図示されているように、CsaConnctable に割り当てられた汎用スロットは、この CsaConnctable に対する応答を受け取るたびにパルス化される。CsaConnectable のための SyncHandle に応じた waitFor...()により、この CsaConnectable に到来するいかなる応答でも待てるようになる。

【０１４６】登録に基づく CsaRemotes および CsaConnectables の汎用スロットの生成は、プロセススタートアップ中に bcs オブジェクトにおいてイネーブル／ディスエーブルにすることができる。

【０１４７】コンシューマ接続

この場合も汎用スロットは bcs オブジェクトによって、それに CsaRemote が登録されるたびに生成される。この汎用スロットは、もっぱら新たに登録されたCsaRemote に対してのみ割り当てられる。CsaRemote オブジェクトは、必要であれば bcs からの SynchHandle のクエリーを行うことができる。

【０１４８】図２０には、CsaRemote オブジェクトと汎用スロットと bcs オブジェクトとの間の相互接続について示されている。図示されているように、CsaRemote オブジェクトに割り当てられた汎用スロットは、CsaRemote によりアップデートが受け取られるたびに（プッシュモード）、あるいはこの CsaRemote により getValue() コールが成功するたびに（プルモード）、パルス化されることになる。

【０１４９】登録に基づく CsaRemotes および CsaConnectables のための汎用スロットの生成は、プロセススタートアップ中に bcs のオブジェクトにおいてイネーブル／ディスエーブルにすることができる。

【０１５０】コントロールインタフェース

汎用スロットにおけるメソッドの一時停止／再開はサポートされていない。

【０１５１】シグナリングあり状態

bcs オブジェクトにより受け取られ CsaConnctable へ伝送される各々の応答によって、汎用スロットがパルス化される。

【０１５２】図２１には、CsaConnctable のシグナリングあり状態が示されている。

【０１５３】CsaRemote スロットについてシグナリングはオペレーションモード（プッシュ／プル）に依存することになる：

・プルモード

bcs オブジェクトは、getValue() コールが CsaRemote オブジェクトに対して成功するたびに、プルモードにおいて この CsaRemodte のための汎用スロットがパルス化される。

【０１５４】図２２には、プルモードにおける CsaRemote のシグナリングあり状態が示されている。

【０１５５】・プッシュモード

bcs オブジェクトは、コールバックがアクティブにされる前にこの CsaRemoteのために受け取られた各イベント（アップデート）により汎用スロットがパルス化（セット／リセット）されることになる。

【０１５６】図２３には、プッシュモードにおける CsaRemote のシグナリングあり状態が示されている。

【０１５７】CsaConnectable / CsaRemote がだめにされると、その汎用スロット状態は無効となり、つまりその後はもはや waitFor...() コールにおいて利用できなくなる。この場合、あらゆる論理的な組み合わせにおけるイベントを待っているいかなる waitFor...() コールも中断され、相応の CsaConnectable / CsaRemote が除去された時点でエラーが戻されることになる。

【０１５８】フック

汎用スロットについてフックは設けられていない。

【０１５９】当業者であればいかなる変形実施形態も提案できるであろうが、本発明の範囲は請求の範囲によってのみ制約されることを強調したい。

【０１６０】なお、以下に本発明にかかわるソースコードを添付しておく。

【０１６１】

【外１】

```
class CsaSesam
{
   public:

      typedef unsigned long      SynchHandleType;

      struct BookInfo
      {
         void (*beforeFunc) (void*);
         void *arg1;
         void (*afterFunc) (void*);
         void *arg2;
         void (*beforeCb) (void*);
         void *arg3;
         void (*afterCb) (void*);
         void *arg4;
      };

      enum DynSlotStateType

                { Initial = 256,
                  Queued,
                  Started,
                  Finished,
                  CbActive,
                  Idle
                };

      struct SlotInfo
      {
         char                    myIdString[20];
         DynSlotStateType        mySlotState;
         char                    myCntlState;
      };

      enum CbModeType

                { CbSyncMainDispatch = 256,
                  CbAsyncMainDispatch_SyncSesamDispatch,
                  CbAsyncMainDispatch_AsyncSesamDispatch
                };

      enum RetValModeType

                { KeepRetVal = 256,
                  DismissRetVal
                };

      enum LogicalCombinationType

                { LogAnd = 256,
                  LogOr
                };

      enum TimerModeType

                { TimerSync = 256,
                  TimerAsync
                };

      enum ExceptionNameType

                { Exception1 = 256,
                  Exception2
```

【0162】 【外2】

```
        });

static CssSesam* sesam(void);

bool info (const SynchHandleType& theSh,
           SlotInfo& theInfo);

bool suspend (const SynchHandleType& theSh);

bool resume (const SynchHandleType& theSh);

bool remove (const SynchHandleType& theSh);

bool waitForMultipleObjects (int theNum,
                             const SynchHandleType *const theSh,
                             LogicalCombinationType theCombMode,
                             SynchHandleType *const theSsh = 0,
                             unsigned long theTimeout = 0);


bool activate (void* (*const theFunc) (void*),
               void *const theArg,
               SynchHandleType& theSh,
               void (*const theCb) (void*),
               const char *const theName = "",
               CbModeType = CbAsyncMainDispatch_SyncSesamDispatch,
               const HookInfo *const theHooks = 0);

bool activate (void* (*const theFunc) (void*),
               void *const theArg,
               SynchHandleType& theSh,
               const char *const theName = "",
               RetValModeType = DismissRetVal,
               const HookInfo *const theHooks = 0);

bool getRetVal (const SynchHandleType& theSh,
                void **const theRetVal);

bool isCancelled (void);

bool addTimer (unsigned long theInterval,
               unsigned long theDelay,
               void (*const theCb) (void*),
               void *const theArg,
               TimerModeType theMode = TimerSync,
               const char *const theName = "",
               SynchHandleType *const theSh = 0,
               const HookInfo *const theHooks = 0);

bool registerExceptionHandler (ExceptionNameType theName,
                               void (*const theCb) (ExceptionNameType),
                               CbModeType theMode
                               = CbAsyncMainDispatch_SyncSesamDispatch,
                               const HookInfo *const theHooks = 0);


bool remove (ExceptionNameType theName,
             void (*const theCb) (ExceptionNameType),
             CbModeType theMode
                               = CbAsyncMainDispatch_SyncSesamDispatch);

bool createGenericSlot (SynchHandleType& theSh,
                        const char *const theName = "");

bool set (const SynchHandleType& theSh);

bool reset (const SynchHandleType& theSh);
```

【0163】 【外3】

```
    bool pulse (const SynchHandleType& theSh);

    void dump (void);

protected:

    CseSesam(ACE_Reactor *theMainDispatcher = ACE_Service_Config::reactor());

private:

    ~CseSesam();

    void destroy (void);

    ACE_Thread_Manager                          *myThreadManager;
    ACE_Thread_Manager                          *myThreadManagerInternal;

    ACE_Map_Manager <SynchHandleType,
                     CseSesamSlot*,
                     ACE_Null_Mutex>            *myMapMgr;

    int                                         myMapMgrSize;

    ACE_Reactor                                 *myMainDispatcher;
    short                                       myMainDispatcherFlag;
    int                                         myDispatcherThrId;

    ACE_Reactor                                 *mySesamDispatcher;
    int                                         myDispatcherThrId1;

    ACE_Reactor                                 *mySignalDispatcher;
    int                                         myDispatcherThrId2;

    CseSesamThrTermHandler                      *myThrTermHandlerPtr;

    bool                                        myInitFlag;

    ACE_RW_Mutex                                myRWLock;

    ACE_Message_Queue <ACE_MT_SYNCH>            myMQ;
};
```

【図面の簡単な説明】

【図１】オブジェクト指向プログラムのメインコンポーネントを示す図である。

【図２】オブジェクト指向プラットフォームの継承ハイアラーキの実例を示す図である。

【図３】ハードウェアおよびソフトウェアの環境を示す図である。

【図４】ＳＥＳＡＭに関する環境を示す概略図である。

【図５】ＳＥＳＡＭの基本コンポーネントを示す図である。

【図６】ユーザ関数を非同期で実行する原理を示す図である。

【図７】ダイナミックスロットの遷移状態を示す図である。

【図８】ワンショットタイマに関するアクティビティのシーケンスを示す図である。

【図９】インターバルタイマに関するアクティビティのシーケンスを示す図である。

【図１０】インターバルタイマに関するアクティビティのシーケンスを示す図である。

【図１１】別のコールバックの呼び出しに基づくタイマコールバック呼び出しを示す図である。

【図１２】タイマスロットの一時停止の様子を示す図である。

【図１３】インターバルタイマスロットのシグナリングあり状態を示す図である。

【図１４】呼び出しモードにより分けられたユーザ定義システムの例外ハンドラのスタックを示す図である。

【図１５】同じ例外のための複数の例外ハンドラをパラレルに実行する様子を示す図である。

【図１６】ディスパッチャを介して呼び出すことで例外ハンドラルーチンをシリアル化するプロセスを示す図である。

【図１７】コールバックモード CbAsyncMainDispatch_AsyncSesamDispatch を利用したときの異なる例外のための複数の例外ハンドラをパラレルに実行する様子を示す図である。

【図１８】例外スロットのシグナリングあり状態を示す図である。

【図１９】CsaConnectable と汎用スロットと bcs オブジェクトとの間の相互接続の様子を示す図である。

【図２０】CsaRemote と汎用スロットと bcs オブジェクトの間の相互接続の様子を示す図である。

【図２１】CsaConnectable のシグナリングあり状態を示す図である。

【図２２】プルモードにおける CsaRemote のシグナリングあり状態を示す図である。

【図２３】プッシュモードにおける CsaRemote のシグナリングあり状態を示す図である。

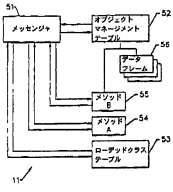
【符号の説明】

１１　オブジェクト指向コンピューティング環境

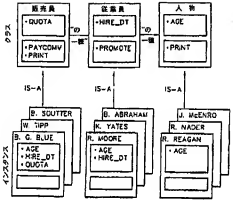
１２　コンピュータプラットフォーム

13　コンピュータハードウェアユニット
14　CPU
15　メインメモリ
16　I／Oインタフェース
21　ディスプレイ端末
22　入力機器
23　不揮発性データ記憶装置
24　プリンタ
100　オペレーティングプロセス
102　アプリケーションレベル
104　メインディスパッチャ
108　SESAMディスパッチャ
110　SESAMシグナリングディスパッチャ

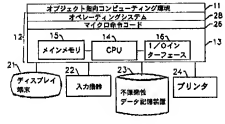
【図1】



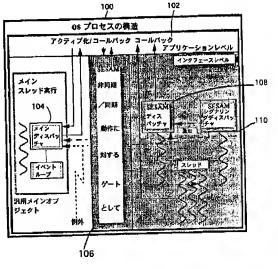
【図2】



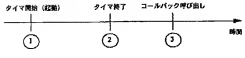
【図3】



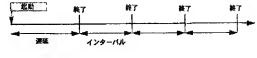
【図4】



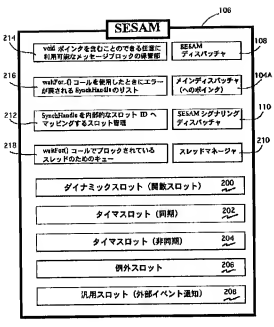
【図9】



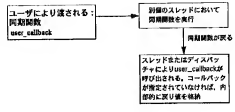
【図10】



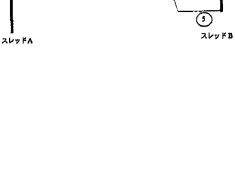
【図5】


SESAM
106
214
void ポインタを含むことのできる任意に利用可能なメッセージブロックの管理部
SESAM ディスパッチャ
108
216
wait(or..)コールを使用したときにエラーが戻される SyncHandle のリスト
メインディスパッチャ（へのポインタ）
104A
212
SyncHandle を内部的なスロット ID へマッピングするスロット管理
SESAM シグナリング ディスパッチャ
110
218
wait(at) コールでブロックされているスレッドのためのキュー
スレッドマネージャ
210
ダイナミックスロット（同期スロット）
200
タイマスロット（同期）
202
タイマスロット（非同期）
204
例外スロット
206
汎用スロット（外部イベント通知）
208

【図6】


ユーザにより渡される：同期関数 user_callback
別個のスレッドにおいて同期関数を実行
同期関数が戻る
スレッドまたはディスパッチャにより user_callback が呼び出される。コールバックが指定されていなければ、内部的に戻り値を保持

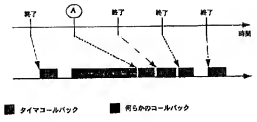
【図8】


関数とコールバックの実行依頼
1
7
SESAM スロット
生成
2
wait(or
アンブロック
3
4
5
スレッドA
スレッドB

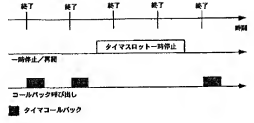
【図7】

300
初　期
1
activate()
302
キューイング
activate()
310
アイドル
7
2
スレッドの生成と開始
6
戻り値獲得
コールバック実行
5
開　始
CbActive
304
308
3
4
関数が戻る
コールバック開始
終　了
306

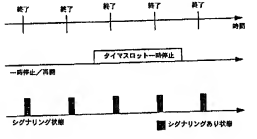
【図11】



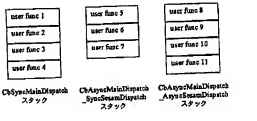
【図12】



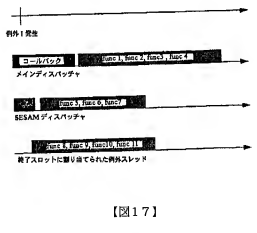
【図13】



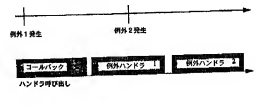
【図14】



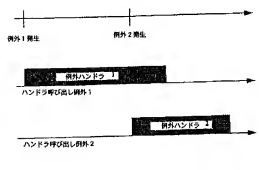
【図15】



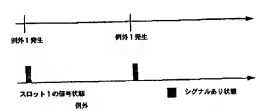
【図16】



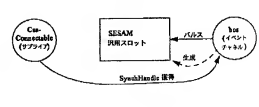
【図17】



【図18】

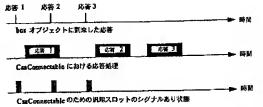


【図19】

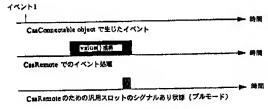


【図20】

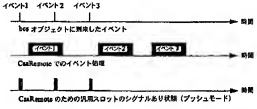
【図21】



【図22】



【図23】



フロントページの続き

(72)発明者　ディートリッヒ　クエール
ドイツ連邦共和国　エアランゲン　ニュルンベルガー　シュトラーセ　83